ARCHIVOS XML

En este tema, trabajaremos con archivos XML. Para refrescar la memoria, los XML (lenguaje de marcas extensible) es un fichero de texto simple de metalenguaje extensible que consta de diferentes etiquetas que contienen los datos.

Un XML se compone de la declaración del XML:

<?xml versión= "1.0" encoding= "UTF-8"?>

En esta parte se declara el XML, la versión del documento y se define el *encoding* que se utilizará en el fichero.

Debemos definir esta línea como nuestra primera línea de cualquier fichero XML. Los campos *versión* y *encoding* deben estructurarse en este orden para considerarse una estructura correcta.

De todos modos, las declaraciones son opcionales, pero si se establece *encoding* se deberá añadir también la declaración *versión*. La versión nos servirá para indicar en qué momento se realizó el documento y seguir una evolución del estándar si se modifica el archivo en un futuro.

El *encoding*, nos permitirá definir qué caracteres vamos a utilizar. Por defecto, utilizaremos UTF-8.

A continuación, añadiremos el cuerpo del XML, que es la parte más importante del fichero. Este documento adquiere una estructura de árbol, compuesto por un elemento raíz o principal dentro del cual añadiremos el resto de los elementos.

<raíz>

<tronco>

<rama1></rama1>

<rama2></rama2>

</tronco>

</raíz>

Podemos entender el concepto **metalenguaje** como el código que utilizaremos para describir la información que queremos transmitir en un fichero XML. Es un lenguaje especializado para describir nuestro lenguaje natural en código: mediante símbolos, iremos representando la estructura de lo que se quiera representar.

Como vemos en este ejemplo de cuerpo, podemos ver que se estructura por un elemento padre "raíz" del cual se desprenden diferentes hijos, en este caso, el elemento "tronco", o subhijos, que serían los elementos "rama". Podrá tener tantos hijos como sea necesario, pero el elemento padre no se podrá repetir.

Dentro de cada etiqueta, se podrá encontrar la información de cada elemento.

Para finalizar, nuestro XML debería parecerse a algo más o menos así:

```
<?xml version="1.0" encoding="UTF-8"?>
<coches>
<coche>
<marca>Seat</marca>
<modelo>Ibiza</modelo>
<color>rojo</color>
<matriculacion>2019</matriculacion>
</coche>
<coche>
<marca>Ford</marca>
<modelo>Focus</modelo>
<color>gris</color>
<matriculacion>2014</matriculacion>
</coche>
</coches>
```

Este tipo de fichero se utiliza en muchos programas para comunicarse los unos con los otros y transportar diferentes datos entre ellos.

Analizadores sintácticos (*parser* DOM y SAX) y vinculación (binding JAXB)

El soporte XML en Java tiene diferentes API que nos permitirán trabajar con la información de estos archivos.

Un analizador sintáctico es básicamente un objeto que permitirá leer la información del XML y acceder a ella para extraerla.

Un **parser** es un analizador sintáctico para XML que se encarga de verificar que la estructura de ese fichero de texto es correcta.

Hay diferentes tipos que nos ofrecen diferentes ventajas frente a otros.

DOM

La API DOM se caracteriza por ser un analizador basado en modelos de carga de documentos con estructuras en árbol, el cual guarda en memoria la información del XML. Permite a los programas y a los *scripts* acceder y actualizar dinámicamente el contenido, la estructura y el estilo de un documento.

Entre las características principales de este analizador podemos destacar que nos permitirá tener los datos en orden, navegar por ellos bidireccionalmente (es decir poder recorrer una relación en ambos sentidos: desde A puedes llegar a B y desde B puedes volver a A) y disponer de una API de lectura y escritura de datos, así como también la manipulación del fichero XML.

Lo único negativo que cabe destacar es que el *parser* DOM tiene un procesado de información bastante lento, lo que provoca que consuma y ocupe mucho espacio en memoria del programa al cargar o tratar el fichero XML.

Ejemplo DOM:

Como se ve en el código de la página siguiente, para empezar, declaramos un fichero XML. Tendremos que indicarle la ruta del fichero. El fichero tendrá esta estructura con estos datos:

```
<?xml version= "1.0" encoding="UTF-8"?>
<coches>
    <coche>
          <marca>Seat</marca>
          <modelo>Ibiza</modelo>
          <color>rojo</color>
          <matriculacion>2019</matricula-
cion>
    </coche>
</coches>
```

Si nos fijamos, en esta ocasión le pasamos la ruta relativa del fichero. Este fichero estará en una carpeta creada dentro de nuestro proyecto, dentro de la carpeta *main/resources/xml*, aunque puede usarse la ruta absoluta, solo tendréis que cambiar esa ruta por la de vuestro fichero.

A continuación, es necesario crear una nueva instancia del objeto DOM que cargará el archivo XML en la memoria. Esta nueva instancia será la clase *DocumentBuilderFactory*, que nos va a permitir obtener un analizador sintáctico que produce la jerarquía de objetos DOM a partir de documentos XML.

Una vez creada la instancia de esta clase, necesitaremos definir una clase *DocumentBuilder*. Esta es necesaria para posteriormente definir el documento que nos permitirá parsear el XML.

```
try{
//Indicaremos la ruta del fichero xml
//Src es el nombre raiz de nuestro proyecto, main es la primera
capreta, resources la siguiente, dentro de xml encontraremos el fichero
//Esta es la ruta relativa.
    File arxXml = new File("src/main/resources/xml/coches.xml");

    //Creamos los objetos que nos permitiran leer el fichero
    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
    DocumentBuilder db = dbf.newDocumentBuilder();
    //Le pasamos el XML
    Document doc = db.parse(arxXml);
    doc.getDocumentElement().normalize();
    System.out.println("Elemento raiz:" + doc.getDocumentElement().
    getNodeName());
    NodeList nodeList = doc.getElementsByTagName("coche");
    //Creamos un bucle para leer los datos del xml y los mostramos en
la consola
    for (int itr = 0; itr < nodeList.getLength(); itr++)  {
          Node node = nodeList.item(itr);

          if (node.getNodeType() == Node.ELEMENT_NODE){
                Element eElement = (Element) node;
                System.out.println("Marca: "+ eElement.
                getElementsByTagName("marca").item(0).
                getTextContent());
                System.out.println("Modelo: "+ eElement.
                getElementsByTagName("modelo").item(0).
                getTextContent());
                System.out.println("Color: "+ eElement.
                getElementsByTagName("color").item(0).
                getTextContent());
                System.out.println("Matriculacion: "+ eElement.
                getElementsByTagName("matriculacion").item(0).
                getTextContent());
          }
    }
} catch (Exception e) {
    e.printStackTrace();
}
```

Después crearemos una nueva instancia de la clase *Document*, que nos permitirá almacenar nuestro documento XML.

La clase *Document* representa un documento XML y nos proporcionará el acceso al contenido del documento XML.

Seguidamente, necesitaremos obtener el nodo raíz a través del método *getDocumentElement()*, así obtendremos los datos de esa etiqueta.

A continuación, tenemos que detectar cuántos elementos contiene el XML, es decir, cuántos nodos hay definidos.

Para ello, utilizamos el método *getElementByTagName()* al cual le pasaremos el nombre del nodo que queremos sustraer, en este caso, "coche". Para almacenar los datos obtenidos, definiremos el *NodeList*, que almacenará todos los datos encontrados haciendo la llamada al método *getElementByTagName()*.

Si necesitamos acceder a todos los nodos desde el inicio del fichero, podemos llamar recursivamente a este método: *getChildElement()*.

En el ejemplo vemos también que definimos la clase *Node*, esta nos servirá para asignar todos los datos de cada elemento "coche" encontrado en el XML.

Para obtener el valor del texto, podemos usar el método *getElementByTextValue()* para buscar un nodo por su valor, y para acceder a los datos de los atributos, *getElementByTagName()* junto con el método *getAttribute()*.

SAX

Otro modo de acceder a los datos de un XML es con la API SAX (Simple API for XML). Esta librería se caracteriza por estar basada en eventos para parsear los datos. Los métodos son **callbacks**; el parser los va llamando **en orden** conforme avanza por el documento XML, línea por línea. No hay "retorno" de un método a otro; hay **nuevo evento → nueva llamada**.

Si la comparamos con el acceso DOM, es mucho más eficiente porque no carga en memoria todo el árbol del fichero, sino que lee los ficheros usando una función para informar al cliente de la estructura del documento. Por tanto, es una opción mucho más rápida de analizar el contenido de los ficheros XML que y no consumirá tantos recursos de la aplicación que usemos.

El inconveniente más destacable es que es un poco más complicada de utilizar que el resto de *parsers*, y que no ofrece navegación bidireccional, analiza los datos del fichero de forma secuencial, empezando al inicio del XML y acabando al cierre final.

Ejemplo SAX:

Para la implementación de SAX, primero necesitaremos tener un fichero XML. Usaremos el mismo fichero XML sobre coches que en el ejemplo DOM.

Debido a su mayor complejidad para poder explicar mejor cómo estructurar una implementación de un ejemplo con SAX, vamos a desglosar el ejemplo por partes para poder explicarlo mejor.

En primer lugar, necesitamos definir la clase *SAXParserFactory* y crear una nueva instancia:

```
SAXParserFactory factory = SAXParserFactory.newInstance();
SAXParser saxParser = factory.newSAXParser();
```

Esta clase es la API encargada de proporcionar un SAX parser. Seguidamente, se deberá crear un SAX parser que se obtendrá gracias a la llamada del método *newSaxParser().*

```
SAXParserFactory factory = SAXParserFactory.newInstance();
SAXParser saxParser = factory.newSAXParser();

SaxHelper handler = new SaxHelper();
saxParser.parse("src/main/resources/xml/coches.xml", handler);
```

Como vemos en el ejemplo, para poder parsear el XML deberemos llamar al método *parse()*, al que se necesita pasarle por parámetro la ruta del fichero XML (para ruta absoluta recuerda añadir file\\ al inicio de la ruta para evitar errores) y un handler.

Lo que hará es llamar a una clase auxiliar de apoyo que veremos cómo se crea a continuación.

Un ***handler*** es una clase auxiliar que servirá para realizar los diferentes pasos de extracción de datos del XML. Deberemos crear una clase auxiliar llamada *SaxHelper* que extenderá de *DefaultHandler* y que nos proporcionará la implementación por defecto de los métodos necesarios para realizar el *parser*.

Qué hace SAX hasta aquí:

- SAX trabaja en **streaming**: lee el XML **de izquierda a derecha** y va lanzando **eventos** (inicio de elemento, texto, fin de elemento…).
- Tú **no construyes** un árbol (como DOM). Solo **reaccionas** a esos eventos en el handler.

Para entender mejor en qué consiste, vamos a ver su implementación:

```
public class SaxHelper extends DefaultHandler{

    boolean esMarca = false;
    boolean esModelo = false;
    boolean esColor = false;
    boolean esMatriculacion = false;
}
```

La clase se llamará en este caso “SaxHelper”, pero podemos asignarle el nombre que más nos convenga. Este método se llamará cuando se encuentre el principio de un elemento, este deberá extender siempre de la clase *DefaultHandler*.

De este modo, nos podremos beneficiar de los métodos por defecto. Si no se implementa así, no nos funcionará el *parser* con SAX.

A continuación, debemos crear tantas variables como elementos y atributos tenga nuestro XML. En nuestro caso, tenemos cuatro elementos, por lo que creamos cuatro booleanos con sus nombres asociados.

Estas variables nos servirán para saber si el elemento que estamos comprobando corresponde al que queremos encontrar. Por tanto, tendremos uno para marca, modelo, color y matriculación.

Si alguno de nuestros elementos tuviera atributos, por ejemplo <matriculación pais="ES">2019</matriculacion> habría que declarar un String paísMatrícula e inicializarlo en null para posteriormente poder acceder a ese atributo.

Nuestra clase auxiliar, para funcionar correctamente, tendrá que implementar un método *startElement*, el cual se encargará de buscar los diferentes elementos y atributos de nuestro XML. Este método se llamará cuando se encuentre el principio de un elemento.

```
public void startElement(String uri, String localName, String elemen-
tos, Attributes atributos) throws SAXException {
    System.out.println("Inicio del elemento :" + elementos);

    switch (elementos) {
          case "marca":
                esMarca = true;
                break;
          case "modelo":
                esModelo = true;
                break;
          case "color":
                esColor = true;
                break;
          case "matriculacion":
                esMatriculacion = true;
                break;
          default:
                break;
          }
}
```

Si nos fijamos, el método *startElement* se encarga de ir comprobando que el *string* "elementos" contenga uno de nuestros elementos del XML. Cuando encuentre uno de ellos, establecerá el valor *true* a la variable que corresponda según la etiqueta.

Si alguno de nuestros elementos tuviera atributos, por ejemplo <matriculación pais="ES">2019</matriculacion> usaríamos atributos.getValue("pais") para obtener "ES":

paisMatricula= atributos.getValue("pais")

Si el atributo no existiera, devuelve null.

A continuación, deberemos implementar el método *characters()*.

```
public void characters(char ch[], int inicio, int length) throws SAX-
Exception {
    if (esMarca) {
          System.out.println("Marca: " + new String(ch, inicio,
length));
          esMarca = false;
          return;
    }

    if (esModelo) {
          System.out.println("Modelo: " + new String(ch, inicio,
length));
          esModelo = false;
          return;
    }

    if (esColor) {
          System.out.println("Color: " + new String(ch, inicio,
length));
          esColor = false;
          return;
    }

    if (esMatriculacion) {
          System.out.println("Matriculacion: " + new String(ch, ini-
cio, length));
          esMatriculacion = false;
          return;
    }
}
```

Este método será llamado cuando se encuentra texto en el XML. Es un método que recibe por parámetro:

• Un *array* de tipo *char* que contendrá todos los caracteres de nuestro XML.

• Un *int* inicio que contendrá un *int* que indicará la posición en la que tiene que empezar a leer el *array char* anterior.

• Un *int lenght* que indicará el número de caracteres que tenemos que usar del *array* de caracteres.

El método se encargará de comprobar qué elemento debe imprimir por consola, ya que en el método anterior hemos verificado qué elemento estábamos comprobando. Si la propiedad booleana correspondiente a ese elemento del XML es *true*, imprimirá por consola el elemento y el valor correspondiente, y volverá a setear a *false*.

Esto se hace para indicar que ya no está en uso ese elemento.

Para finalizar con este *helper*, necesitaremos implementar el método *endElement()*. El método se llama cuando encuentra el final de un elemento.

En este ejemplo, solo se imprimirá por consola el valor del elemento e indicará que hemos terminado con este bloque del XML.

```
public void endElement(String uri, String localName, String elementos)
throws SAXException {
    System.out.println("Fin del elemento: " + elementos);
}
```

Este método recibe por parámetro un *string* con la URI, otro con *localName* y otro con el elemento XML finalizado.

En este caso, solo tendrá valor el *string* "elementos" porque nuestro XML es más sencillo y el resto de campos (uri y localName) están vacíos. Cabe decir que solo se rellenarán cuando el XML contenga esa información.

Excepciones en SAX

SAXException/SAXParseException por parseo

IOException por I/O

ParserConfigurationException al construir el parser.

En resumen:

El parser SAX tiene un **bucle interno** que va leyendo el XML byte a byte.

Cuando encuentra algo, **dispara un evento** y **llama** al método correspondiente de tu DefaultHandler:

- <etiqueta> → llama startElement(...)
- texto → llama characters(...)
- </etiqueta> → llama endElement(...)

Luego **continúa leyendo** el flujo. Si lo siguiente que aparece es otra etiqueta de apertura, **volverá a invocar** startElement(...).

No es que *retorne* de un método endElement() a uno startElement() sino que **el parser vuelve a "notificar"** el evento siguiente.

El parser seguirá, a grandes rasgos, una secuencia de callbacks de este tipo:

1. startElement("coches")
2. startElement("coche")
3. startElement("marca") → activa esMarca=true
4. characters("Seat") → imprime Marca y apaga esMarca
5. endElement("marca")
6. startElement("modelo") → esModelo=true
7. characters("Ibiza") → imprime Modelo y apaga esModelo
8. endElement("modelo")
9. startElement("color") → esColor=true
10. characters("rojo") → imprime Color y apaga esColor
11. endElement("color")
12. startElement("matriculacion") → esMatriculacion=true
13. characters("2019") → imprime Matriculacion y apaga esMatriculacion
14. endElement("matriculacion")
15. endElement("coche")
16. endElement("coches")
17. endDocument()

Este es un ejemplo muy simplificado para observar el funcionamiento básico de SAX.

JAXB

Una de las posibilidades más interesantes que nos ofrece Java es la posibilidad de convertir un XML en un objeto Java. La librería JAXB (Java XML API Binding) nos ofrece esa posibilidad. Está incluida dentro de la JDK de Java, por lo que no es necesario importar librerías.

El ***data binding*** es el concepto para definir la voluntad de transformar un fichero XML (o cualquier otro fichero) en un objeto Java.

Algunas de las características principales de esta API son que nos permite navegar por el documento en ambas direcciones y permite tanto la manipulación de ficheros XML como su conversión a Java de manera muy simple.

Unmarshal: XML → objeto Java.

Marshal: objeto Java → XML.

Como aspecto negativo, cabe destacar que no permite tratar un fichero XML si este no es válido, por tanto, tenemos que validarlo antes de poder transformarlo a un objeto Java.

Ejemplo JAXB:

```
String xml = "<?xml version= \"1.0\" encoding=\"UTF-8\"?>"
+ "<coches><coche><marca>Seat</marca><modelo>Ibiza</modelo>" +
"<color>rojo</color><matriculacion>2019</matriculacion>"
+ "</coche></coches>";

JAXBContext jaxbContext;

try {
jaxbContext = JAXBContext.newInstance(Coche.class);
// Este objeto se encargará de la transformación a objeto Java que le
indiquemos.
    Unmarshaller jaxbUnmarshaller = jaxbContext.createUnmarshaller();
    Coche coche = (Coche) jaxbUnmarshaller.unmarshal(new StringRead-
er(xml));
    System.out.println(coche);
} catch (JAXBException e) {
    e.printStackTrace();
}
```

Tal y como podemos ver en el ejemplo, la API JAXB se encarga de coger los datos del XML y convertirlos en el objeto que le indiquemos. El objeto debe tener los mismos atributos que el XML, si no, se lanzará la excepción.

¿Cuándo usar cada API?

Editar árbol, XPath, conservar formato: **DOM**.

Streaming / muy grande / parcial: **SAX**

Objetos ↔ XML (modelo estable, comodidad): **JAXB**.

XML a JSON

En Java existen una infinidad de librerías que podemos añadir a nuestro proyecto para poder transformar nuestros XML a cualquier otro formato. En este apartado, explicaremos cómo transformar nuestros ficheros XML a un formato JSON.

JSON (Javascript Objecto Notation) es un tipo de archivo de formato de texto derivado de Javascript, bastante ligero y que almacena información estructurada. Es fácil de interpretar y generar, y se utiliza para transferir información entre un cliente y un servidor.

Los archivos JSON siguen una estructura basada en definición de objetos, asignando un atributo y un valor. Un fichero JSON es capaz de definir seis tipos de valores: *string*, número, objeto, *arrays*, *true*, *false* o *null*. Veamos un ejemplo:

Para definir el JSON:

Se abren y cierran corchetes.

Los objetos se declaran entre comillas y los diferentes objetos se separan con una coma.

El nombre y el valor de cada pareja van separados entre dos puntos.

Cada objeto se considera un ***string;*** en cambio, los valores de los atributos pueden ser de cada tipo permitido nombrado anteriormente.

```
public static String XML_PRUEBA = "<coche><id>1</id><modelo>Ibiza</mo-
delo><marca>seat</marca></coche>";
try {
    //Creamos el objeto que nos ayudara a convertir el XML en JSON
      JSONObject json = XML.toJSONObject(XML_PRUEBA);
      //Identamos el json, le damos formato
      String jsonFormatado = json.toString();
      System.out.println(jsonFormatado);
} catch (JSONException je) {
      System.out.println(je.toString());
}
```

Para esta conversión tenemos que añadir la librería Jackson a nuestro proyecto.

Esta librería nos permitirá realizar la transformación de XML a JSON en muy pocas líneas de código.